7-25-1983

A Software Prototype for the IBM Series/1 Remote Intelligent Printers

Thomas Weber

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A SOFTWARE PROTOTYPE FOR THE IBM
SERIES/1 REMOTE INTELLIGENT PRINTERS

by

Thomas J. Weber

A Thesis Submitted to the Faculty of the
College of Computer Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Computer Science

Rochester Institute of Technology
One Lomb Memorial Drive
Rochester, New York

May 16, 1983
A SOFTWARE PROTOTYPE FOR THE IBM
SERIES/1 REMOTE INTELLIGENT PRINTERS

by

Thomas J. Weber

This thesis was prepared under the direction of the candidate's thesis advisors: Mrs. Mary Ann Dvonch, Dr. John Ellis, and Mr. Guy Johnson. It was submitted to the faculty of the College of Computer Science and Technology and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

SUPERVISORY COMMITTEE:

Professor John Ellis, Thesis Advisor

Professor Mary Ann Dvonch, Committee Member

Professor Guy Johnson, Committee Member

[Signature]

July 25, 1983

Date
STATEMENT OF THESIS REPRODUCTION

I, Thomas J. Weber, hereby deny permission to the Wallace Memorial Library, of Rochester Institute of Technology, to reproduce my thesis in whole or in part without written permission from myself. Any reproduction that is allowed by myself shall not be for commercial use or profit. I can be contacted at the following address.
ACKNOWLEDGEMENTS

I would like to thank the faculty of the Rochester Institute of Technology, College of Computer Science and Technology, particularly Dr. Peter Anderson, Mrs. Mary Ann Dvonch, Dr. John Ellis, and Mr. Guy Johnson. These professors openly supported the development of this thesis under the employment of IBM. Without their flexibility, this opportunity would have not been possible.

I would like to thank three employees of International Business Machines Corporation: Mr. Edward Kloster, Mr. Thomas Rowlett, and Mr. Randy Forlenza. These men invested considerable time and resources so that I could work for IBM on this project.

Through the joint efforts of RIT and IBM this thesis was made possible. I want to express my gratitude to RIT and IBM for their cooperative effort and overall support.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgements</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis Abstract</td>
<td>viii</td>
</tr>
<tr>
<td>Introduction to the Series/1 Project</td>
<td>1</td>
</tr>
<tr>
<td>a. project coordination</td>
<td></td>
</tr>
<tr>
<td>b. problem statement</td>
<td></td>
</tr>
<tr>
<td>c. assumptions, and limitations</td>
<td></td>
</tr>
<tr>
<td>d. design, development, and test plans</td>
<td></td>
</tr>
<tr>
<td>The Original Series/1 System Hardware and Software</td>
<td>7</td>
</tr>
<tr>
<td>a. 4955 processor</td>
<td></td>
</tr>
<tr>
<td>b. FPCA communications controller</td>
<td></td>
</tr>
<tr>
<td>c. 4975 printer series</td>
<td></td>
</tr>
<tr>
<td>d. EPSON printer</td>
<td></td>
</tr>
<tr>
<td>e. Event Driven Executive (EDX)</td>
<td></td>
</tr>
<tr>
<td>f. Event Driven Language (EDL)</td>
<td></td>
</tr>
<tr>
<td>Conceptual and Theoretical Development of the New Software</td>
<td>19</td>
</tr>
<tr>
<td>a. IOSASCII assembly module conception</td>
<td></td>
</tr>
<tr>
<td>b. GENASCII CCB system-generation macro conception</td>
<td></td>
</tr>
<tr>
<td>c. model of the EIA RS232C electrical interface</td>
<td></td>
</tr>
<tr>
<td>d. finite-state machines of IOSASCII basic states</td>
<td></td>
</tr>
<tr>
<td>e. finite-state machine implementation in IOSASCII</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion ................................................................. 41

a. accomplishments, problems, and short-comings
b. additional work that could be done
c. effectiveness of Series/1 micro-based hardware components
d. related future extensions
e. academic and professional milestones

Side Thoughts and Suggestions from this Project ............... 47

a. avenue for thesis development
b. representative IBM hardware

Bibliography ....................................................................... 50

Glossary of Acronyms, Terms, and Phrases ......................... 52

Appendices:

a. Series/1 hardware configuration ...................................... A
b. IOSASCII flowcharts .................................................... B
c. IOSASCII assembly listing .............................................. C
d. GENASCII macro listing ............................................... D
e. EDL system generation listing ......................................... E
f. EDL test case listings .................................................... F
# TABLE OF FIGURES

1. EDX I/O Hierarchy Diagrams ........................................... 21

2. EIA RS232C Electrical Interface Model ............................ 27

3. IOSASCII Finite State Machine Diagrams ........................... 29

4. IOSASCII FSM Level Implementation Diagrams ...................... 38
ABSTRACT

Over the period of December 1982 to May 1983, a software prototype to support the IBM 4975 model 01A printer was developed. The host computer was the IBM Series/1 minicomputer. The printer controller used was the Feature Programmable Multi-Line Adapter. Both the controller and printer contain one or more microprocessors that do some processing for the Series/1. With the RS232C electrical interface, the printer has remote capabilities. The electrical interface performs data transmission that is serial, asynchronous, and full duplex. This thesis serves as an application showing how intelligent printers and controllers can be utilized effectively on the Series/1.
INTRODUCTION TO THE SERIES/1 PROJECT

a. Project Coordination

This thesis project was the result of a joint venture between Rochester Institute of Technology (RIT) and International Business Machines Corporation (IBM). In the history of the RIT College of Computer Science and Technology (CS&T), such an arrangement is unprecedented. First of all, the thesis was developed off-campus at Boca Raton, Florida. Second, it was a full-time effort under the employment of IBM. This means that a competitive salary was paid on a non-exempt basis as a type of co-operative employment, only on the graduate level. Since RIT is not a research and development institution, the pursuit of such an arrangement is recommended to my colleagues, for both the hands-on experience and the materialistic benefits.

b. Problem Statement

The major milestone of the thesis was to develop software to coordinate the activities of a remote printer, its controller, and the supporting mini-
computer. The primary printer for which development was done was the IBM 4975 model 01A ASCII Printer. This is an 80 character per second (CPS) printer with remote capabilities. Real-time software control of the printer and its controller needed to be developed. The software needed to ensure that the printer was never over-run with data. Also, the state, or status of the remote printer needed to be monitored at all times.

The primary controller was the IBM 2095/2096 Feature Programmable Multi-Line Controller and Adapter (FPCA). This controller will support up to 8 remote 4975 printers. The data communications between the controller adapter card and the printer is based on the EIA RS232C electrical interface. This interface uses a pacing feature in which the printer can notify its adapter card that the printer's internal buffer is nearly full of data and can accept no more. When the buffer is partially emptied by the printer, it notifies the adapter card to re-start data transmission. The controller of the adapter cards is responsible for cycle-stealing data from main memory to the specific adapter card for transmission.

The host computer used was the IBM Series/1 minicomputer. This machine was designed primarily for data communications, real-time control, and sensor-based I/O. It is commonly used for distributed processing, networking, and even traditional data processing applications.
The Series/1 sees the FPCA through a single hardware address. It is up to the software to recognize which of the adapter cards needs servicing. Event-driven interrupts from the printer and the controller are processed through the software that was developed. It is possible for event-driven interrupts to occur virtually at any time.

c. Assumptions and Limitations

The technical foundation of this project is based on a set of assumptions. These assumptions must be guaranteed for the implementation of the project.

First, the data communications medium between the remote printer and the Series/1 must have little or no interference. The data communications protocol is only capable of recognizing one inverted bit per byte transmitted. That is, parity checking is all that is supported by the hardware. Once more, with the detection of a parity error by the remote printer, the printer must print the damaged character: it has no capability to request re-transmission of the data. Thus, this parity check system implies a low noise data communications medium.

Second, with this kind of transmission error processing, transmission
speed must be slow enough to ensure that a noise spike causes as little damage as possible. This is a trade-off situation. A low line speed will result in less data transmission errors, while a higher line speed will result in greater throughput. The 4975 printer hardware supports line speeds of 1200BPS and 4800BPS. A local, switched, leased line should be able to successfully carry 1200BPS transmissions.

With a low noise medium, and a moderate transmission speed, some data parity errors will occur on occasion. Thus, this particular printer should be used where critical processes are independent of the data printed.

Certain system-dependent limitations also exist. These are from the original design of the Series/1 hardware and EDX operating system. First, main memory storage was assumed to be expensive and therefore limited. With a 16 bit machine and an operating system (OS) that performs no paging, the entire OS must be resident in 64K bytes of memory. This excludes any utilities that are loaded on request, usually into other 64K partitions.

Second, the simple element of development time was limited originally to four months. With an extra month's effort, the entire scope of the project appears to be far in excess of four months with the available resources.
d. Design, Development, and Test Plans

Project design was highlighted by several finite state machines (FSM) of the FPCA driver module. These FSM's depict the state transitions that occur and must be supported on three basic levels. Development began with the software that supported primitive level one, and then proceeded to the second and third primitive level functions. The FSM diagrams were an important design tool and have been included in Figure 3. These FSM's are discussed in detail in the 'Theoretical and Conceptual Development' section of this report.

Project development was an iterative process. First, an initial set of flow charts was developed. The charts were repeatedly studied and then refined. Each successive set of charts was more detailed and resolved more problems than the previous set. When sufficient flow-charting confidence was reached, the software was coded. The final set is included in Appendix B.

The initial driver that was coded only supported the basic controller/printer data communications. This driver allowed the basic coding errors to be found. These were errors dealing with wrong assembly language instruction testing conditions, resetting posted events, and incorrect directions sent to the printer's controller.
The next stage of development is still on-going. This is the process of resolving errors that are solely a function of processing speed. For example, if the communications code is executed slowly, no errors in transmission may occur. With selected stop-on-addresses, certain errors may occur, while at full processor speed, different errors occur.

The general test plan was to sequentially test each FSM primitive level. Sequential testing ensured that a specific problem would not be duplicated in additional code before it was resolved in a subordinate level. The initial states that were coded included only those needed for the basic data communications protocol. This is the majority of the support, but it is the smallest stand-alone unit that can be tested. Next, routines were added serially to support other functions like special-case error handling, printer-status retrieval, FPCA buffer clearance, controller-end polling, and modem support. Appendices C through F contain the code developed up to the point of the thesis defense. These appendices include the test cases written for monitoring the behavior of the controller/printer pacing algorithm.
a. The 4955 Master Processor

The basic characteristics of the 4955 processor follow. It is a 16-bit, byte-addressable minicomputer processor with 179 microcoded instructions. There are 28 instruction format types, and 25 different address generation methods. The average instruction execution time is 2.65 microseconds, with a microcycle time of 220 nanoseconds. Main storage ranges from 16K bytes to 512K bytes with a storage cycle time of 660 nanoseconds. There are four priority interrupt levels, each with its own set of eight general-purpose registers and status indicators. Level switching can occur by hardware and software control.

With respect to this project, the 4955 processor is the master of the computer system. Subordinate to it is the FPCA controller. The 4955 processor instructs the controller to do work in parallel with the processor or by executing an operate I/O instruction. This instruction contains sufficient information for the controller to independently execute its own microcode on an ATOM 1 microprocessor while the 4955 processor contin-
ues its work. When the controller needs to fetch data for transmission, or store it after reception in main memory, it cycle steals data during 4955 instruction cycles that do not use main memory. The main processor is notified by an interrupt when a requested communications I/O is complete.

b. The FPCA Communication Controller

The Feature Programmable Multi-Line Communication Controller (FPCA) is designed to provide control circuitry for one or two programmable four-line adapter cards. Each adapter card is programmable for serial, synchronous or asynchronous, data communications. The following discussion describes additional features which are also programmable.

Any transmission code can be used with 5, 6, 7, or 8 bits per character. The 7-bit ASCII code was used for the 01A printer. Line change-of-direction (COD) characters can be programmed for half-duplex line reversals or for full-duplex event occurrence signaling. In this project, the latter was true and was used for XON, XOFF, and printer-status-ready event posting during a full-duplex transmission. Parity generation can be set to enabled or disabled. If enabled, then even or odd parity can be specified. Odd parity was programmed for the 01A printer. The number of synchronization characters per synchronous
transmission or the number of start/stop bits per asynchronous transmission can also be specified. The RS232C protocol required the programming of one start bit and one stop bit per asynchronous character transmitted. There are various other programmable features, but they do not apply to this thesis and will not be discussed.

There are ten operate I/O commands that can be requested of the FPCA. Of these, five were used. These are: prepare, device reset, start control, start cycle steal status, and start cycle steal I/O. Prepare, device reset, and start control are used to initialize the FPCA. The prepare command is used to control the interrupt parameters of the addressed adapter card. This includes enabling its interrupt ability and setting the hardware level on which interrupts will occur. The device reset command resets the addressed card and clears any pending interrupt requests. The start control operate I/O places the FPCA in extended mode which supports the environment needed for the full duplex RS232C protocol. These three commands are executed at IPL time to initialize the state of each FPCA adapter card that has a 4975 01A printer attached. To recover from certain fatal communication errors, the device reset and start control commands are re-issued to place a particular FPCA adapter card back in operation.

The start cycle steal status command provides three additional words
of status information regarding the last operate I/O that the FPCA adapter card attempted. This command is used for two purposes. First, it is used to recover from a data communications error. Words one and two of the FPCA status information provide error information regarding why an operate I/O failed. Possible errors are data over-run, adapter card time-out, incorrect command, modem interface error, and others. The second purpose of performing a start cycle steal status operation is to obtain the data transmission residual address. This is the address of the last character that was successfully received or transmitted before data communications halted.

The start cycle steal I/O command is used to start a cycle-stealing operate I/O transmit or receive operation. This command requires the specification of the FPCA adapter card address, the data message buffer address, the message length, and two timer values. When the I/O is executed, two sets of return codes usually result. The first set indicates whether or not the command was accepted by the controller and its subordinate addressed adapter card. This set represents immediate, pre-I/O return codes. If the first return code indicates success, then the second set is forthcoming. The second set indicates the I/O completion code and is presented through an interrupt. These indicate that the I/O was either successfully completed or terminated with an error. Once a cycle steal operate I/O is successfully initiated, a wait is executed which is
posted with the FPCA adapter card I/O completion code in the FPCA interrupt handler.

A recently documented feature of the FPCA is its ability to support full duplex data communications. By placing the FPCA in expanded mode using the set control operate I/O command, any FPCA adapter card can be programmed to continuously receive data and generate an interrupt if specific characters are received. Previous software support simulated such hardware behavior by always having a receive operation outstanding whenever a transmit was not being performed. This simulation was used for half-duplex terminal support by the IOSACCA module.

Expanded mode with continuous receive and attention interrupt capabilities is capable of supporting the EIA RS232C electrical interface. Using the set mode operate I/O, the transmission-on (XON), transmission-off (XOFF), and printer-status-ready (1B) characters are programmed as COD characters. This way, when one of these COD characters is received from the printer for data pacing or printer-status-ready, an interrupt is automatically generated, which invokes servicing for the reported condition. These interrupts are capable of occurring during a transmission and halting it if necessary. Any FPCA adapter card is capable of continuously receiving up to 15 characters, at which time the buffer must be explicitly emptied by a read-adapter-buffer operate I/O.
c. The 4975 Printer Series

During the second half of 1982, a new series of micro-based printers was announced by IBM for the Series/1. This was the 4975 series. They are table-top, wire matrix, bi-directional, serial-impact, EBCDIC printers with 132 character printing carriages. Each printer contains an INTEL micro-processor and its own microcode in read-only memory (ROM). Models 01L and 01R operate at 80CPS and support draft-quality printing only. Models 02L and 02R operate at 160CPS when printing draft-quality and 40CPS when printing correspondence output. The 01R and 02R models have remote capabilities. The intended use of the 4975 series is to meet the Series/1 system needs for a small system printer, a work station printer, or a remote printer. These printers are sold as part of a Series/1 system and cost approximately $4200 each.

There are two electrical interfaces available for the 4975 series. The EIA RS422A interface is for models 01L and 02L supporting local attachment to the Series/1. The EIA RS232C interface is for models 01R and 02R which allows remote use. These 4975 printers use the Multi-Function Attachment (MFA) card as their Series/1 interface. Their method of operation is as follows. First the MFA transmits control information to the printer regarding the message to be printed. This is the data control block (DCB). It contains such information as the main storage data ad-
dress, and message length. The printer then requests that the MFA card send it one print line of data. The MFA then uses the RS232C or RS422A asynchronous protocol to transmit the one print line. The printer then prints that line and requests the next line. The result is a half-duplex technique in which the printer synchronizes the transmission of a print line in which each character is transmitted asynchronously without any pacing control. This method uses the MFA as a slave to the printer. The new support developed for the 4975 01A printer uses the FPCA as the master and the printer as its slave. Appendix A is a diagram showing the attachment of the 01A printer to the Series/1.

The 4975 model 01A ASCII printer belongs to the previously discussed 4975 printer series. There are several internal differences though. First, unlike the other 4975 printers, the 01A is subordinate to its communications controller, the FPCA. Second, it is capable of reporting three conditions: buffer full, buffer emptying, and printer-status-ready. These states are reported asynchronously at any time. Third, its printing speed is only 80CPS while its slowest data reception speed is 120CPS. Since the line speed is always faster than the printing speed, there is an obvious need for data transmission pacing. Fourth, this printer supports the ASCII character set. In fact, this printer could be sold as an Other Equipment Manufacturer (OEM) device due to its rigid conformity to ASCII and EIA RS232C standards.
The 01A printer can be used as a remote printer. Because of this, the printer has the ability of transmitting 17 bytes of printer status information to its FPCA adapter card. These bytes describe conditions like out of forms, loss of power, data framing error, data parity error, and others. Whenever the 01A has status to report, it transmits a hexadecimal '1B' and tries to continue printing. The FPCA adapter card should respond by sending a message requesting the transmission of the 17 byte printer status report.

Another feature of this printer is its data transmission error recovery method. If the printer receives a damaged character, it has no ability to transmit a negative acknowledgement (NACK), to request data re-transmission. The printer simply prints a histogram character in place of the damaged character and then sends printer-status-ready (1B) over the FPCA adapter line. The only possible error recovery is to recognize the printer-status-ready condition, request printer-status-data transmission to the FPCA, analyze the printer status information, recognize that a damaged character has been printed and re-start the software application from the beginning. This is not an acceptable recovery technique, so that the 01A printer must not be employed in situations where occasional loss of data would cause a serious problem.
d. The EPSON Printer

The EPSON printer is a product of the Shinshu Seiki Company in Japan. It is marketed by IBM as the Personal Computer 1 printer, feature number 5152. This printer uses the same RS232C electrical interface seen in the 4975 01A model. The IOSASCII module will also support this printer. The only differences are that the EPSON does not report status, it prints only 80 column output, and it has a 2K byte data reception buffer.

e. The Event Driven Executive

The Event Driven Executive (EDX) is a Series/1 basic supervisor and emulator. The supervisor size ranges from 15K bytes to 35K bytes and must be main-memory resident: there is no paging support. Only utility programs, like the text editor, are loaded from secondary storage, usually in some other storage partition. The supervisor always resides in partition zero. Up to eight partitions of 64K bytes each are possible. The supervisor supports multiple, independent, time-dependent, and/or event driven applications. An event driven application is one in which the occurrence of interrupts controls the application's behavior. The supervisor supports four high level languages. These are BASIC, COBOL, FORTRAN, and PL/I. These languages compile into the Event Driven
Language (EDL) which is again compiled and executed by the EDL emulator.

The EDX supervisor is generated by the user to exactly match the user's hardware configuration. For every device on the Series/1, a device declaration statement is used. These statements generate tailored control blocks for each device. These control blocks contain device description parameters, save areas, and linkage to other control blocks. In the case of this project, every FPCA adapter card has its own control block, called a terminal control block (CCB). A CCB is one type of control block that is used for terminals, printers, the MFA, the FPCA, and other devices. Since the 4975 01A is attached to the Series/1 by the FPCA, only one control block for each adapter card is necessary. There is no control block explicitly for the printer. The CCB used for each adapter card of the FPCA also controls the printer attached to that adapter card. The generation of each CCB occurs at system generation (sys-gen) time. Appendix D contains a listing of the GENASCII macro that generates the CCB. Appendix E contains the resulting macro expansion listing for the supervisor used in this project. This supervisor labels the 4975 01A printer with the logical name ASCIIIPRT. This is the logical name that is used to secure serial ownership of a physical device, use it, and then return it for other applications to use.
f. The Event Driven Language

The Event Driven Emulator supports the Event Driven Language. This is a high-level, macro-instruction language in which EDL instructions are actually translated, or emulated, at execution time. Many of the EDX utilities are written in EDL. Appendix F contains two EDL program examples. When dealing with a machine that was designed with the assumption that primary memory is expensive, EDL is an excellent language. This is because each EDL instruction requires an average of only six to eight bytes of storage. The trade-off here is execution time for storage.

EDL instructions go through the following process before they can be executed. First they are compiled. The output is a listing of subroutine calls to the Series/1 assembly language modules that support each EDL instruction. For each branch-and-link (BAL) subroutine call, there is a parameter list following it. This list is generated by the EDL compiler to control the execution of the EDL's assembler module. Thus, a compiled EDL program becomes nothing more than an ordered list of parametrical subroutine calls and a task control block for the EDL program. EDL instructions are executed serially using their support modules which reside in partition zero with the supervisor.

At the EDL level, there is complete device independence. For
example, the user's application program does not need to present any additional information to print data to a terminal, a printer, or a data communications controller. But, before an application program attempts to communicate to a device, it must first obtain exclusive ownership of that device. This is necessary to secure the integrity of the device's CCB and to prevent data interweaving. The EDL ENQT instruction with the device's logical name is necessary to secure device ownership. When an application program is finished with the enqueued resource, it executes a DEQT EDL instruction to free it. The printer test cases in Appendix F contain EDL ENQT and DEQT examples.

For EDL application programs to be device independent, there must be a device dependent module to support the hardware intricacies of each device. For this project, the IOSASCII Series/1 assembly language module provides the device-dependent support for the FPCA and the 4975 01A printer. IOSASCII uses the CCB information from sys-gen time to direct FPCA communications to the 4975 01A printer. Since each FPCA adapter card uses its own CCB, the IOSASCII module is re-entrant. Now, only one copy of the module needs to be storage-resident for any number of attached 4975 01A printers.
CONCEPTUAL AND THEORETICAL DEVELOPMENT

a. IOSASCII Assembly Module Conception

IOSASCII is the software driver module for the 4975 01A printer attached to the FPCA. Its predecessor was IOSACCA, which supported terminals attached to the FPCA. Some code was salvaged from IOSACCA and used in IOSASCII. The modem routine, controller-end polling routine, and the FPCA busy routine were taken from IOSACCA. These routines do not exceed 10% of the IOSASCII module.

During development, IOSASCII was named IOSACCA so that the already existing linkage could be used at sys-gen time for generating test supervisors. Figure 1 shows the three levels of the software hierarchy that links the IOSASCII module into the EDX supervisor. Level one is the user's application program. At this level, the user codes EDL PRINTTEXT and PRINTNUM statements which produce program output. Also coded is a PROGRAM statement which generates a task control block (TCB). The TCB is the primary control block which allows the user's program to be treated as a task by the supervisor.
Level two is managed by the EDXTIO module. This module takes the output from level one and converts it to ASCII strings of characters. These are placed in the terminal control block (CCB) buffer. When the buffer becomes full, the third level is invoked.

Level three is managed by the IOSTERM module. This module contains common routines for peripherals attached by a general data communications controller. It provides services such as carriage-return/line-feed insertions and setting flags for modem and other control operations that IOSASCII must perform before doing the actual transmission of data. IOSTERM may call IOSASCII several times to accomplish the printing of one line of data on the 4975 01A printer.

Figure 1 also contains a generalized data and control flow diagram. This diagram depicts the data flow, control flow, and linkage for terminal and printer I/O. The three levels of the EDX supervisor are also depicted in this diagram.
The terminal I/O support is organized into the following three layers:

- Instruction interpretation and attention handling
- Code translation, numeric conversion, and buffer management
- Terminal support routines

1 These modules contain translation tables only. They do not contain any executable code.

FIGURE 1
(1 of 3)
EDX PRINTER AND TERMINAL I/O SUPPORT

The diagram on the following page shows the control paths, data flow, and control block connections for the basic printer and terminal support.

(1) Level 1: instruction interpretation and attention handling
   (1a) instruction interpretation entry points (EDXTIO)
   (1b) attention handling entry points

(2) Level 2: buffer management and code translation
   (2a-d) buffer management entry points
   (2e) conversion between binary and ASCII number representations

(3) Level 3: terminal and printer support routines
   (3a) data transfer initiation
   (3b) new-line support
   (3c) interrupt servicing
   (3d) terminal support entry points

(4) hard-copy function for 4978/4979 display

FIGURE 1
   (2 of 3)
FIGURE 1

(3 of 3)
b. GENASCII CCB Sys-Gen Macro Conception

GENASCII is a macro that generates the device-dependent section of the CCB, which precedes the common section of the CCB. The device-dependent section and the common section make up the complete CCB control block. The CCB dependent and common sections are listed at the end of Appendix C.

The device dependent section contains data needed specifically for the FPCA with a 4975 01A printer attached. This section includes COD counters, fetch-printer-status chained DCB's, a read-adapter-buffer DCB, a fetch-FPCA-status DCB, a set mode DCB, and an expand mode DCB. These are accessed as negative offsets from the CCB common section. On entry to the IOSASCII module, the base register contains the address of the common section. By using the CCB data section (DSECT) listed at the end of IOSASCII, the negative portion is accessed. The CCB DSECT is a storage map that gives the relative offsets of all CCB fields, which when added to the base register contents, yields the partition zero physical address of the desired CCB field. Only 15% of GENASCII is new, the remaining 85% was taken from the GENACCA macro which doubles with the IOSACCA module.

The CCB common section is standard for all devices that use CCB's.
Other devices like disks and tape units do not use CCB's. They have their own control block format. Most of the common section is generated by the TERMINAL macro which calls the GENASCII macro to generate the FPCA-dependent area. During development, GENASCII was named GENACCA so that the already existing linkage could be used at sys-gen time for generating test supervisors.

c. Model of the EIA RS232C Electrical Interface

The data communications protocol between the FPCA adapter card and the printer is the EIA RS232C electrical interface. This is a serial, asynchronous, full-duplex interface. Data can be transmitted in two directions at any time. The FPCA can transmit text strings of any length to the printer and the printer can send COD characters to the FPCA adapter card. Since text strings are transmitted at a line speed that exceeds the printing speed, it is necessary for XON/XOFF pacing to break-up text strings into smaller strings at paced intervals.

Figure 2 depicts two diagrams representing data transmission. The first is a series of messages M0 through Mn that are to be transmitted for printing. Each of these messages are in turn broken into smaller messages by the XON/XOFF pacing. These sub-messages are represented by
m0 through mn. The exact length of the sub-messages or the time at which they are subdivided can not be predicted. Message subdivision is controlled by the hardware and is a function of printer speed, transmission rate, and system work-load.

The second diagram of Figure 2 depicts the timing scheme of message transmission. In this diagram, each message to be transmitted is subdivided into as many as n sub-messages. These sub-messages are transmitted with time-lapses between them. Each time lapse is associated with the last sub-message transmitted. Time lapses represent the period of time that elapses between the event occurrences of the reception of an XOFF and an XON printer control character. There is a brief period of time between the reception of XOFF and the actual termination of data transmission. There is also another brief period of time between the reception of the XON character and the actual re-start of transmission. These are included in the time lapse period. If all of the transmission time intervals and associated time-lapse intervals are summed, the result is the total time per message transmission. This models the transmission scheme used by the EIA RS232C electrical interface as a function of time.

There also is data transmission in the opposite direction. This is the transmission of printer COD characters which represent printer events. These transmissions do not require pacing because they never exceed two
EIA RS232C TIMING MODEL

MESSAGE TRANSMISSION SCHEME

\[ M = \text{one complete message to be transmitted} \]
\[ m = \text{a subset of } M \]
\[ n = \text{the number of message subdivisions} \]
\[ t = \text{transmission time interval} \]
\[ l = \text{unknown transmission halted time-lapse} \]

FIGURE 2
characters in length and occur seldom enough so that the FPCA 15-byte buffer is never over-run.

d. Finite-State Machines of the IOSASCII Basic States

A finite state machine is a graphical representation of the flow of state transitions that occur in the machine being modeled. In this case, the IOSASCII software module is the machine. The state transitions are event-driven by interrupts presented by the FPCA and the 01A printer.

There are three primitive levels that the IOSASCII module can be in. These are: initial program load to initial program load with no transmit active, transmit initiated, and transmit active and other non-transmit I/O executing. Transition between the first and second primitive levels occurs when an application program performs an EDL PRINTEXT or PRINT-NUM statement. Transition between the second and third states is event driven by one of many possible interrupt conditions. Within each of the three primitive levels is a family of subordinate states. Figure 3 contains the FSM diagrams.

When the Series/1 is first powered-up and EDX is loaded and executed by the initial program load (IPL) process, primitive level one is entered.
FIGURE 3 (3 of 3)

Note 1: All state changes are software-driven on level 3.

Primitive level 2: Transmit active with non-transmit I/O executing.
During the IPL process IOSASCII is invoked to initialize each adapter card of the FPCA that supports an 01A printer. The initialization process performs an adapter-card reset, programs COD characters with a set mode operate I/O, activates the full-duplex capabilities with the expanded mode operation, and returns to the supervisor. If these operations are successful, the particular FPCA adapter card is now functional, otherwise it enters an error state. Assuming success, an event-driven state change will occur next. Every three minutes the 01A printer transmits a re-start transmission (XON) character. XON's will continue to be sent until primitive level two is invoked. The reception of an XON may be immediately followed by a printer-status-ready (1B), causing a state change to the '1B' state. Both the XON and/or '1B' states result in nothing more than counting the occurrence of the interrupt. When primitive level two is invoked, the XON count is reset and the '1B' count causes the invocation of the fetch-printer-status routine to determine what is wrong. The constant reception of XON and '1B' COD characters will eventually fill up the FPCA adapter card buffer and cause the buffer-full state to be entered. This state may require the execution of the read-adapter-buffer routine in IOSASCII to clear the buffer for future reception of printer COD characters.

Primitive level one can also be entered by an application program doing nothing but requesting the printer's status. The request will result in
the fetching of the printer's status which is stored in the CCB for the user's application. If printer status is unsuccessfully fetched, the error state is entered. The error state can be cleared only by the re-initialization of the particular FPCA adapter card.

When an application program enqueues an 01A printer and performs an EDL print statement, primitive level two is entered. Before a transmission to the printer is attempted, primitive level one flags indicating FPCA error or 01A printer status outstanding are checked. If they are set, they are serviced and then the transmission is initiated. During the transmission, four interrupt-driven states can occur. These are device end, stop transmission (XOFF), re-start transmission (XON), and operate I/O error. Device end signifies the successful completion of the transmission, in which case the FSM end-state occurs and primitive level one becomes active again. XOFF signifies that the 01A printer buffer is full and the FPCA must wait for it to clear. XON signifies that the 01A printer buffer is now clear and the FPCA can resume data transmission. The error state results from a pre-I/O fatal communications error which sets a flag, invokes the end state and then returns to primitive level one.

If the XOFF state was entered, then an XON will be presented by the printer when it has buffer space for more data. The reception of XON causes a state change to the XON state. In the XON state, three events
can occur. First, the transmission may be re-started from the point where it left off. This is the expected state. Or, the printer-status-ready state may be entered by a '1B' interrupt, signalling a printer problem. Otherwise, the error state indicating an FPCA problem may be entered. The error state results in the flagging of the pre-I/O fatal communications error and the end state is entered. The end state causes primitive level one to become active.

Primitive level three is entered in one of four ways. First, the occurrence of an XOFF interrupt during a transmission causes level three to be invoked through the FPCA status state. This state results in the execution of a fetch FPCA status operation to obtain the EDX buffer address of the last character transmitted. This address is placed in the transmit DCB and control returns to primitive level two.

Second, primitive level three is entered through the printer status state by the occurrence of a '1B' interrupt during transmission. This state is responsible for fetching the printer's status information and re-starting transmission. If the printer status is successfully obtained, and does not show a serious problem, transmission must now be restarted. Transmission re-initiation occurs by the execution of the XOFF state, the fetch FPCA status state, and the XON state under program control rather than event-driven control. The XOFF state invokes the FPCA status
state and fetches the needed re-start address. Next, the XON state re-starts the transmission process, which causes a return to primitive level two. If the fetch-printer-status state was entered from level one, then only the printer status is fetched and control returns to level one.

Third, when the FPCA continuous receive buffer becomes full, it causes primitive state three to be entered through the buffer full state. This state empties the buffer with the read-adapter-buffer operation and then returns to the level it was invoked from. The buffer full state can be entered from primitive level one or two.

Fourth, the occurrence of an FPCA error during a transmission causes primitive state three to be invoked. The transmission error causes an exception interrupt and the error state is entered. The error state attempts to: fetch the FPCA status information that would describe the error, set the FPCA re-initialization pending flag, and return to the application program with error information available in the application's TCB. It then returns to primitive level two which returns control over to primitive level one.

During the servicing of these four level-three entry states, it is possible for attention interrupts to occur from the printer. If they do, they are simply flagged, returning control to the level three entry state which
was interrupted. Level two is responsible for initiating service for such printer attention interrupts.

These three FSM primitive levels properly manage the states of the IOSASCII module. Their primary function of serializing possibly concurrent events is fully supported. They also serve to pair-up the posting of an event with the routine that was expecting the event to occur. The end result is the proper coordination of both program-driven events as well as interrupt-driven events. Together these events implement the necessary support for the 01A printers.

e. Finite-State Machine Implementation

Each of the three FSM primitive levels are supported by a set of routines in IOSASCII. In some cases, routines are shared by the levels and level state flags are maintained to keep track of the present level. There are two state flags that are used for level maintenance. These flags are called TRANSMIO and OTHERIO. When the flags are off, IOSASCII is in FSM level one. When TRANSMIO is on, IOSASCII is in level two. When both flags are on, IOSASCII is in level three.

IOSASCII is divided into two major sections. One section supports da-
ta transmission. The other section, called IAASCII, handles all FPCA and printer interrupts. These two sections execute on different hardware levels. The data transmission section operates on a hardware level one lower than the interrupt handler. This means that the interrupt handler can interrupt the data transmission section at any time to service an interrupt. These two sections are depicted on pages one and two of Figure 4.

Within these two sections is the support for the three FSM primitive levels. The data transmission section contains primitive levels two and three. Page one of Figure 4 contains two boxed-in areas. The routines in the first box support FSM level two. The WRASCII routine with its subordinate subroutines along with the IOCYCLE routine and its subordinate subroutines make up FSM level two. This is the transmission active level. The routines in the second box of Figure 4 support FSM level three. The subroutines subordinate to the IOCONTUE routine make up FSM level three. This is the transmission active with a non-transmit I/O executing level.

Page two of Figure 4 contains a third boxed-in area, which is the interrupt handler. This is FSM level one. This is the support for the IPL to IPL with no transmit active level. The interrupt handler section also contains portions of code that control the servicing of interrupts depend-
FIGURE 4
(1 of 2)
FIGURE 4

( 2 of 2 )
ing on the primitive level that is active. By testing the level state flags, the interrupt handler knows what action to take given any interrupt condition. So actually, there are code segments in the interrupt handler for each of the three levels, not just level one. Appendix B contains the flow charts for the IOSASCII module. These charts show the setting and resetting of the level state flags so that the interrupt handler knows how to process a given interrupt.
CONCLUSION

a. Accomplishments, Problems, and Short-Comings

Project accomplishments can be measured in two ways. The first way uses academic accomplishments. They include such things as an understanding of the coordination necessary for multiple micro-based devices in a real-time fashion; for multiple micro-based devices in a real-time fashion; understanding the data communications needed to implement the master/slave relationship of the Series/1, the controller, and the printer; and an understanding of how emulated user programs and operating system modules interface with primitive OS modules to accomplish the requested function. The amount of knowledge acquired in these areas has been significant, even extensive in the Series/1 environment.

The second way of measuring project accomplishments is through productivity. That is, to what extent was program support developed that will eventually be marketed as a software product by IBM? Approximately 80 percent of the intended support was developed. These modules are in Appendices C through F.
There were more problems encountered during development than expected. The major problem was the amount of time needed to acquire the hardware and its documentation. These were not secured until two-thirds of the project time had elapsed. Another problem was the difficulty in acquiring definitive answers to specific hardware questions. Many questions were eventually resolved by testing the hardware to observe its behavior. Under these circumstances, the project took 1,100 hours, or 0.6 man years.

Project short-comings are apparent in two areas. The first area is productivity short-comings. With respect to IBM, some disappointment exists because the project was not completely accomplished in the time originally budgeted. The second area deals with the IBM hardware. Basically, minor modifications in the design of the controller and printer could have made the software support simpler and more robust. These dealt with the small FPCA buffer and the printer's inability to completely recover from transmission errors.

b. Additional Work that Could be Done

As seen in the development and test plan discussion, approximately 80 percent of the total support was developed. The remaining code to be
added or modified will become apparent as the support is completely de-
bugged. Modules that have not been tested include the multiple printer
controller-end polling routine, the modem support routine, the fetch prin-
ter status routine, and the attention-interrupt exception routine.

c. Effectiveness of Series/1 Micro-Based hardware Components

Traditionally, the concept of distributed processing applies to the use
of remote minicomputers to off-load processing from the main-frame com-
puter. With the constantly decreasing cost of microprocessor chips, intel-
ligence has been distributed on the minicomputer level as well. Now
controllers and their attached peripherals also do specialized processing
to reduce the workload of the minicomputer, which may or may not be su-
bordinate to a main-frame. This is the Series/1 concept. Each layer of
intelligence distributes processing commands to subordinate levels to pro-
vide independent execution of a specific function. Layered intelligence al-

dows a particular level to continue its processing without the need to
continually govern the execution of a function by a subordinate level.
Now a particular level of intelligence is free to do the processing it is best
for, rather than constantly communicating to monitor and govern an at-
tached machine. The particular level can simply send a command directive
and then continue whatever work it has available. The subordinate level
will then notify its master level when the processing it was told to do is complete.

This thesis project is an example of distributed processing on the mini-computer level. The FPCA controller, adapter cards, and the 4975 printer are all capable of independently providing processing for the main 4955 processor. Of course, adding levels of intelligence to independently execute specified functions also adds additional overall system complexity. There now is a need for a data communications protocol between the levels to provide both handshaking and ensure data accuracy. This project used the EIA RS232C protocol. Also, individual levels of intelligence need their own programming to guide their specified processing. For the Series/1 FPCA controller and printer, level programming was implemented in microcode.

In an effort to thoroughly understand the communications protocol and its implementation in this project, several finite-state machines were developed. These machines accurately model the communications environment on three basic levels. The first basic level monitors the behavior of the remote printer in an idling state, where no printing occurs. The second basic level monitors the printer's behavior while it is printing data. The third level models the maintenance processes that may occur intermittently during data transmissions. These three FSM sub-levels work to-
gether to accurately demonstrate system needs for remote printer support. They were the basis for software development and proved to be an effective modeling tool. In this project, the FSM's also provided a comprehensive, common-base model for project discussion with RIT faculty.

d. Related Future Extensions

Research on the trends in microprocessor applications, reveals that there are two extensions on the horizon that pertain to the environment of this thesis. First, printers and vector-graphic type terminals will become interchangeable. That is, the same software support needed to support a graphics terminal will also support a printer. This will be accomplished by building even more processing power into the printer. An example would be drawing a circle on a printer. The printer would be sent the circle's color, radius, and focus point. The printer would internally calculate the points that lie on the circle and print it. For a given printer technology, which has a certain price/performance coefficient, this kind of printer improvement does not push the printing technology to the limits that degrade reliability.

The second extension applies to peripheral-controller technology. The
Series/1 is an example of a minicomputer that will support other equipment manufacturer's (OEM) devices that use standard electrical interfaces. This idea of developing compatible peripherals will be extended one step further. With the advent of further electrical data bus standards, controllers will also become OEM devices. Traditionally, controllers are considered an integral part of a computer system. This will gradually give-way to the computer system OEM component approach. But, computer equipment vendors that stress system packages over component sales may slow the evolution of OEM controllers.

e. Academic and Professional Milestones

The milestone of this thesis project was to show how distributed processing in the Series/1 minicomputer environment can be effective. This thesis provides an example of how built-in microprocessors can perform processing functions, and data communications coordination that would otherwise be done by the main processor. We have seen how the Series/1 minicomputer needs to only initiate a command to an intelligent controller which in turn can manage an intelligent remote printer. Thus, through the vehicle of this thesis, the intention of demonstrating the effectiveness of distributed processing in the Series/1 minicomputer environment has been accomplished.
SIDE THOUGHTS AND SUGGESTIONS FROM THIS PROJECT

a. Avenue for Thesis Development

The development path taken for this thesis has some advantages. As seen in the introduction already, corporate employment for thesis development yields hands-on experience with hardware that would otherwise be unavailable. Considerable interaction with another group of technical experts presents knowledge and development techniques that were not demonstrated at RIT. Hand and hand with the non-exempt wage that was paid was the goal-oriented development attitude towards productivity. With the work experience gained, many new employment avenues became available at IBM, one of which was a position in Robotic communications, which has been accepted. Another advantage included the reimbursement for all relocation costs as well as thesis defense travel costs.

There were of course some disadvantages also. First of all, the resulting matrix management for project development required double approvals, checkpoints, and deadlines. In this case, three RIT professors, two IBM managers, and an IBM technical expert; thus, six approvals were obtained for the thesis proposal and each thesis status
The disapproval of any one of these people required the re-working of the material, typically the contents of a status report. There also was a considerable amount of development pressure that resulted from the realization of the true scope of the project. Simply stated, the project became too large. If the advantages and disadvantages are weighed against each other, the advantages would be found to be somewhat greater.

b. IBM Representative Hardware

One of the difficulties encountered at the start of this project was the transition of coding style from RIT Digital Equipment Corporation (DEC) assembly language to IBM Series/1 assembly language. The design of these two instruction-set types is considerably different and requires the utilization of different programming techniques.

Since IBM is the second largest employer of RIT CS&T students for both co-operative employment as well as full-time employment, there is unquestionably sufficient grounds for RIT to pursue the acquisition of representative IBM hardware. Arrangements have been negotiated with IBM and several major universities throughout the United States for code development for IBM, at no cost to the universities. This type of ar-
rangement may be possible at RIT as well and should be readily pursued.
BIBLIOGRAPHY


IBM 4975 Printer Model 01A (7-Bit Code) Description, GA34-1595.


IBM Series/1 4955 Processor and Processor Features Description, GA34-0021.

IBM Series/1 Communications Features Description, GA18-2033.

IBM Series/1 Digest, G360-0061.

IBM Series/1 Event Driven Executive Communications and Terminal Applications Guide, SC34-1705.

IBM Series/1 Event Driven Executive Internal Design, LY34-0212.

IBM Series/1 Event Driven Executive Language Reference, SC34-1706.
IBM Series/1 Event Driven Executive Macro Assembler Reference, GC34-0317.

IBM Series/1 Event Driven Executive Messages and Codes, SC34-0403.

IBM Series/1 Event Driven Executive Operator's Reference, SC34-1703.

IBM Series/1 Event Driven Executive Program Preparation Guide, SC34-1704.

IBM Series/1 Event Driven Executive System Guide, SC34-1702.

IBM Series/1 Macro Assembler Reference Summary, SX34-0128.

IBM Series/1 Small Computer Concept, SH30-0237.


<table>
<thead>
<tr>
<th>GLOSSARY OF ACRONYMS, TERMS, AND PHRASES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1B</strong></td>
</tr>
<tr>
<td><strong>2095/2096</strong></td>
</tr>
<tr>
<td><strong>4955</strong></td>
</tr>
<tr>
<td><strong>4975 01A</strong></td>
</tr>
<tr>
<td><strong>ACCA</strong></td>
</tr>
<tr>
<td><strong>ASCII</strong></td>
</tr>
<tr>
<td><strong>BAL</strong></td>
</tr>
<tr>
<td><strong>BPS</strong></td>
</tr>
<tr>
<td><strong>CC</strong></td>
</tr>
</tbody>
</table>
CCB terminal control block. A control block that defines a device's characteristics, provides temporary storage, and contains links to other system control blocks.

COD change of direction. A character used with an ACCA device to indicate a reverse in the direction of data movement. With respect to this thesis, a COD represents the occurrence of an external event.

CPS characters per second. Similar to BPS except it applies to characters, not bits.

DEQT EDL dequeue instruction. The DEQT instruction is used to return a device to the pool of available devices once it is no longer needed by some process. A DEQT instruction is preceded by an ENQT instruction.

EBCDIC extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded control and graphic characters.

ECB event control block. A control block used to record the status of an event. It is often used to synchronize the execution of tasks.

EDL event driven language. The language for input to the EDX compiler. The output is interpreted by the EDX emulator.

EDX event driven executive. A multi-tasking, multi-programming supervisor for the Series/1 minicomputer.

EIA Electronical Industries Association. A organization dedicated to research in the electrical industries.

ENQT EDL enqueue instruction. The ENQT instruction is used to obtain exclusive use of a single-user device. This guarantees that no data interweaving or CCB damage occurs.

EPSON IBM Personal Computer I printer. A printer which utilizes the EIA RS232C electrical interface. It is a table-top, wire-matrix printer. This was the secondary printer used for this thesis project.
FPCA feature programmable multi-line communications controller. A data communications controller and up to two attachment cards to support up to 8 devices. It is a programmable, cycle-stealing device that provides synchronous or asynchronous data communications.

FSM finite state machine. A FSM is a graphical model depicting the various states and the transitional state paths taken by some process.

GENASCII FPCA sys-gen macro. This macro is responsible for generating a portion of a CCB for a device that is attached by way of an FPCA adapter card. GENASCII generates the CCB portion that is specifically needed for the 4975 01A printer.

IEEE Institute of Electrical and Electronical Engineers. An organization dedicated to research in the electrical discipline.

IOSASCII FPCA driver module. IOSASCII is the module that supports the attachment of 4975 01A printers to an FPCA card. All device dependent functions for the FPCA and the 01A printer are handled by IOSASCII.

I/O input/output.

IPL initial program load. This is the process of loading and executing a fresh copy of the EDX operating system from disk.

ISB interrupt status byte. An 8-bit data byte which indicates the type of event which has just occurred.

K multiplication factor. K implies that the immediately preceding number needs to be multiplied by 1024 to yield the actual total number of bytes.

MFA multi-function adapter. A data communications controller capable of emulating an FPCA on one of its adapter cards. The remaining adapter cards use the EIA RS422A electrical interface.
OEM

other equipment manufacturer. OEM is a term used in reference to computer equipment manufacturers who market computer peripherals that can be interfaced to computer systems manufactured by another organization.

ROM

read-only memory. This is a type of computer memory that can contain both executable code and data, but it can only be read. No data can be written to it.

RS232C

an EIA electrical interface. This is the electrical interface between the 4975 printer and the FPCA adapter card. It is asynchronous, serial, and uses XON/XOFF pacing to prevent printer buffer overflow.

RS422A

an EIA electrical interface. This is an MFA electrical interface used by other models of the 4975 printer series.

sys-gen

system generation. The process of coding an EDL program which describes the hardware configuration of a Series/1 and assembling it to produce an operating system nucleus.

XOFF

transmission off. A COD character used to indicate that transmission is to be terminated immediately, typically due to a full buffer.

XON

transmission on. A COD character used to indicate that transmission is to be restarted, typically due to an emptying printer buffer.
APPENDIX A

Series/1 Hardware Configuration Diagram
SERIES/1 HARDWARE CONFIGURATION

4955 PROCESSOR

CONSOLE
MAIN MEMORY

OTHER
I/O
ATTACHMENTS

SYSTEM CRT

FPCA CONTROLLER

SYSTEM PRINTER

... MODEM ......>>

4975 01A REMOTE PRINTER
APPENDIX B

IOSASCII Flowcharts
TRANSMIT SET-UP ROUTINE

WRASCII

is The FPCA initialization flag on?

A

B

Did the user request printer status?

N

Y

initialize Transmit DCB timers

is this an FPCA initialization call?

N

Y

is this a single character transmit?

N

Y

set-up DCB for a single character transmit

IODELAY

go set-up to do an operate IO

go perform the operate IO

return to the supervisor

Increment printer status requested counter

MODEM

is this a modem initialization call?

N

Y

return to the supervisor

CONNECTION

DCC
TRANSMIT SET-UP ROUTINE

A

- IORESET
  - go do a device reset
  - IOSETUP
  - go set-up To do an operate IO
    - ISOSETMOD
    - go do SETMODE Operate IO

  - was The SETmode Successful ?
    - Y
      - ISOSETUP
      - go set-up To do an operate IO
    - N
      - ISOEXPAND
      - go do expand mode IO

- ISOSETUP
  - go set-up post operate IO completion codes
  - return to the Supervisor

B

(2 of 2)
EXECUTE OPERATE I/O ROUTINES

IOSETUP

- reset event code, reset error code, set controller end pending flag, and disable interrupts

RETURN

IORESET

- do a device reset, operate I/O

RETURN
EXECUTE OPERATE I/O ROUTINES

**Diagram:**

1. **IOSETMOD**
   - Execute operate I/O set mode

2. **IS**
   - The controller or adapter busy?
   - **Y**: Go wait on completion
   - **N**: Did previous reset I/O complete?
     - **Y**: IOSTESTCC
     - **N**: Continue

execute cycle steal operate I/O
(Transmit or modem operation)

is
The controller or adapter line busy?

Y
N

IOTESTCC

busy

go wait on completion
EXECUTE OPERATE I/O ROUTINES

IOCYCLE

IS

Are printer status information waiting?

YES

PRSTATUS

NO

IONOPRST

IOTESTCC

WAS THE I/O SUCCESSFUL?

NO

reset the Transmit I/O outstanding flag

YES

C

go get the printer's status
CONTINUE OPERATE I/O Routines

1. D
   - Is this an XON interrupt?
     - Yes: IOMODP, go prepare to do an operate I/O
     - No: N
   - Is this a printer escape interrupt?
     - Yes: go get the printer status information
     - No: N

2. IOCONTINUE

3. Was the status fetch successful?
   - Yes: Y
     - Reset printer status request flag
     - I/O checkpoint
   - No: N

4. Did the user request the status?
   - Yes: Y
     - Return to the supervisor
   - No: N

5. I/O OFF
OPERATE I/O ERROR ROUTINE

IOCHKERR

STATUS

go get the
FPCA status
information

Save I/O
error ISB
and CS's for
user

was
This a
modem operation
that failed?

Y

MODEMRRTN

N

IOSRETN
**RETURN TO SUPERVISOR**

**ROUTINE**

- IOSRETURN
  - was the I/O operation successful?
    - Yes: set FPCA re-initialization flag
    - No: return to the supervisor

**FETCH FPCA STATUS**

**ROUTINE**

- STATUS
  - go set-up to do an operate I/O
  - execute a read cycle steal status I/O
  - go do post-I/O processing
  - return
PRINTER STATUS Routine

PRSTATUS

I0SETUP

go set-up to do an operate I/O

execute a cycle steal transmit to request printer to send status

execute a cycle steal receive to receive printer status

FINISHIO

go do post-I/O processing

reset printer status request counter

is a transmit I/O outstanding?

Y

set the XOFF outstanding flag to restart transmission

N

return
POST-I/O PROCESSING ROUTINE

1. FINISHIO
2. The controller or attachment busy?
   a. Yes: Busy
      i. Go wait for completion
      j. Return to I/O routine to retry
   b. No: N
3. Reset controller end pending flag
4. Set non-Transmit I/O end pending flag
5. Enable interrupts
6. Was I/O successfully initiated?
   a. Yes: Wait for I/O to complete
   b. No: N
   c. IERROR
7. ISB/CC
8. Return
CONTROLLER OR ATTACHMENT BUSY ROUTINE

1. BUSY
2. Enable interrupts
3. Wait for an interrupt to occur
4. Disable interrupts
5. Reset posted ECB
6. Return
**INTERRUPT HANDLER ROUTINE**

```
IAASCI

is this a controller end?

Y: Poll FPGA lines to find which one has completed its I/O and post it

N: get the ISB and CC for this interrupt

did an attention interrupt occur?

Y: return to the supervisor

N: is there a non-transmit I/O outstanding?

Y: reset non-transmit I/O outstanding flag

N: reset transmit I/O outstanding flag

IAPOSTIT
```
INCREMENT BUFFER COUNT DUE TO C0D RECEIVED

XOFF INTERRUPT?

Y \rightarrow INCREMENT XOFF RECEIVED COUNTER

N \rightarrow XON INTERRUPT?

Y \rightarrow INCREMENT XON RECEIVED COUNTER

N \rightarrow POINTER STATUS READY INTERRUPT?

Y \rightarrow INCREMENT POINTED STATUS READY COUNTER

N \rightarrow RETURN TO THE SUPERVISOR

RETURN TO THE SUPERVISOR

COUNT 2ND BYTE OF PAR STATUS MESSAGE IN BUFFER

INCREMENT POINTED STATUS READY COUNTER

ONE ALREADY RECEIVED?

Y \rightarrow RETURN TO THE SUPERVISOR

N \rightarrow IS A TRANSFER I/O OUTSTANDING?

Y \rightarrow RETURN TO THE SUPERVISOR

N \rightarrow RETURN TO THE SUPERVISOR
**INTERRUPT HANDLER ROUTINE** (3 of 3)

- **F**
  - Is a non-transmit I/O also outstanding?
    - **N**
      - Save the ISB and CDS
    - **Y**
      - IAPOSTIT
  - **Y**
    - Did an error occur?
      - **N**
        - Post the wait (event occurrence)
      - **Y**
        - Set FPCA re-initialization flag
  - **Return to the supervisor**

- **Is this an attention interrupt?**
  - **N**
    - Return to the supervisor
APPENDIX C

IOSASCII Assembly Listing
Prepara 10 DP OPERATE IO

ija 10 INS

set data buffer address in DCB

SET MESSAG Blaze iin DCC

--RS--

mrcomni f

--RS--

transmit from buffer address

set buffer address key in DCD word 0

position key for DCD placement

get buffer address key

zero message size

--RS--

mrcomni f

--RS--

line control routine

it good? go to switched

b message

get is permitted

6 to 10 size

codes in the range of-

check for condition

cod 6, size

pos for initialization, set return

check for null transmit initialize

reset res

--RS--

mrcomni f

--RS--

use transmit control word to see what action is necessary:

--RS--

mrcomni f

--RS--

set message length

set address of transmit area

--RS--

mrcomni f

--RS--

set up for a single character transmit:

character transmit

--RS--

mrcomni f

--RS--

jp wrccmb

--RS--

mrcomni f

--RS--

if negative, it is a single-

--RS--

mrcomni f

--RS--

see if the message is only 1 byte long:

--RS--

branch if mode control

check null transmit initialization

if different word

--RS--

mrcomni f

--RS--

see if this is a control operation or a regular transmit operation:

--RS--

mawi transmit, accibd

--RS--

smart source statement
APPENDIX D

GENASCII Macro Listing
APPENDIX E

System Generation Listing
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(0)</td>
<td>Address of device ID</td>
</tr>
<tr>
<td>A(0)</td>
<td>Address of terminal address</td>
</tr>
<tr>
<td>A(1)</td>
<td>Address of external code name</td>
</tr>
<tr>
<td>A(2)</td>
<td>Address of initialization link to terminal</td>
</tr>
<tr>
<td>A(3)</td>
<td>Address of descriptor of previous link</td>
</tr>
<tr>
<td>A(4)</td>
<td>Address of serial number of device</td>
</tr>
<tr>
<td>A(5)</td>
<td>Address of terminal number</td>
</tr>
<tr>
<td>A(6)</td>
<td>Address of terminal name</td>
</tr>
<tr>
<td>A(7)</td>
<td>Address of maximum line size</td>
</tr>
<tr>
<td>A(8)</td>
<td>Address of maximum buffer size</td>
</tr>
<tr>
<td>A(9)</td>
<td>Address of current line position (initial)</td>
</tr>
<tr>
<td>A(10)</td>
<td>Address of current line position (final)</td>
</tr>
<tr>
<td>A(11)</td>
<td>Address of line size (final)</td>
</tr>
<tr>
<td>A(12)</td>
<td>Address of current line size</td>
</tr>
<tr>
<td>A(13)</td>
<td>Address of current page size</td>
</tr>
<tr>
<td>A(14)</td>
<td>Address of top of working area (initial)</td>
</tr>
<tr>
<td>A(15)</td>
<td>Address of top of working area (final)</td>
</tr>
<tr>
<td>A(16)</td>
<td>Address of flags (final)</td>
</tr>
<tr>
<td>A(17)</td>
<td>Address of flags (initial)</td>
</tr>
<tr>
<td>A(18)</td>
<td>Address of control area module</td>
</tr>
<tr>
<td>A(19)</td>
<td>Address of control area block</td>
</tr>
<tr>
<td>A(20)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(21)</td>
<td>Address of control area block (initial)</td>
</tr>
<tr>
<td>A(22)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(23)</td>
<td>Address of control area block (initial)</td>
</tr>
<tr>
<td>A(24)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(25)</td>
<td>Address of control area block (initial)</td>
</tr>
<tr>
<td>A(26)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(27)</td>
<td>Address of control area block (initial)</td>
</tr>
<tr>
<td>A(28)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(29)</td>
<td>Address of control area block (initial)</td>
</tr>
<tr>
<td>A(30)</td>
<td>Address of control area block (final)</td>
</tr>
<tr>
<td>A(31)</td>
<td>Address of control area block (initial)</td>
</tr>
</tbody>
</table>

Note: The table above represents the structure of a device in a terminal setup, detailing various addresses and their descriptions.
**ERROR**

Invalid input content. The text is not legible or meaningful as it appears to be a corrupted or scrambled version of code or text. Please provide a clear and readable version of the document for analysis.
APPENDIX F

EDL Test Case Listings
PROGRAM START

DATA SET NAME: TEST3

APRIL 9, 1983

TEST PROGRAM

Tom Weberg

EDX Development

For PCA with 4975 A1 Printer Development

EVENT DRIVEN EXECUTIVE - VERSION 3, MODIFICATION LEVEL 2

Page 1 of 8